SYSTEM AND METHOD FOR COMPUTER SECURITY USING MULTIPLE CAGES

CROSS REFERENCE TO RELATED APPLICATIONS

This application claims priority to co-pending U.S. Patent Application No.

5   09/615,967 (Attorney Docket No. RECOP001) entitled SYSTEM AND METHOD FOR

COMPUTER SECURITY filed July 14, 2000, which is incorporated herein by reference

for all purposes; and co-pending U.S. Patent Application No. 09/616,805 (Attorney

Docket No. RECOP002) entitled SYSTEM AND METHOD FOR GENERATING

FICTITIOUS CONTENT FOR A COMPUTER filed July 14, 2000, which is

10   incorporated herein by reference for all purposes; and co-pending U.S. Patent Application

No. 09/615,891 (Attorney Docket No. RECOP003) entitled SYSTEM AND METHOD

FOR PREVENTING DETECTION OF A SELECTED PROCESS RUNNING ON A

COMPUTER filed July 14, 2000, which is incorporated herein by reference for all

purposes; and co-pending U.S. Patent Application No. 09/616,469 (Attorney Docket No.

15   RECOP004) entitled SYSTEM AND METHOD FOR PREVENTING DETECTION OF

A COMPUTER CONNECTION TO AN EXTERNAL DEVICE filed July 14, 2000,

which is incorporated herein by reference for all purposes.

This application is related to co-pending U.S. Patent Application No.

_____ (Attorney Docket No. RECOP008) entitled SYSTEM AND METHOD

20   FOR ANALYZING LOGFILES filed concurrently herewith, which is incorporated

herein by reference for all purposes.

FIELD OF THE INVENTION

The present invention relates generally to computers. More specifically, a system and method for computer security using multiple cages will be disclosed.

BACKGROUND OF THE INVENTION

5   Computers and networks of computers, such as local area networks (LAN) and wide area networks (WAN), are used by many businesses and other organizations to enable employees and other authorized users to access information, create and edit files, and communicate with one another, such as by e-mail, among other uses. Often, such networks are connected or are capable of being connected to computers that are not part

10   of the network, such as by modem or via the Internet. In such cases, the network becomes vulnerable to attacks by unauthorized users, such as so-called computer "hackers", who may be able to gain unauthorized access to files store on network computers by using ports or connections provided to connect the network to computers outside of the network.

15   One known technique for foiling an attacker seeking to gain unauthorized access to a computer or computer network is a so-called "honey pot." A honey pot, in computer security parlance, is a computer system containing a set of files that are designed to lure a computer hacker or other attacker to access the files, such as by making it seem like the files are particularly important or interesting. Since the honey pot files are typically not

20   actually working files, any activity in the honey pot files is suspicious and an attempt is made to identify and locate any user who accesses or attempts to access the files.

The major shortcoming of the honey pot approach is that by the time the attacker has accessed the honey pot files, the attacker has already gained access to the computer containing the files. The attacker also has access to other files on the same computer, and may be able to access other computers in the same computer network. There is typically no mechanism for restricting the hacker to viewing only the honey pot files.

A second known approach is to provide a deception server. A deception server contains false data. A router or firewall is configured to route suspected attackers to the deception server instead of permitting the suspected attacker to access the real computer system or network.

The major shortcoming of prior art deception servers is that it is relatively easy for attackers to discover they are in a deception server. Among other things, prior art deception servers cannot make it appear to an attacker that the attacker has been allowed on the actual computer or computer network. In addition, deception servers have only a limited number of files, with the result that it is relatively easy to determine that a deception server does not contain the full array of files typically found in a true server, such as a typical business network computer server. With prior art deception servers, it is not practical to have multiple instances (to simulate different hosts) running on a single system, because it is relatively easy to determine that the apparent multiple hosts are in fact running on a single system.

As a result, there is a need for a way to deceive attackers into believing they have gained access to a true computer system or group of systems, without actually allowing

them to gain access to true files. In addition, there is a need for a way to monitor such

attackers, without their knowing, to facilitate efforts to improve security measures and

identify attackers, including automated tools to assist in analyzing logfiles.

SUMMARY OF THE INVENTION

Accordingly, a system and method for computer security are disclosed.

It should be appreciated that the present invention can be implemented in numerous ways, including as a process, an apparatus, a system, a device, a method, or a

5   computer readable medium such as a computer readable storage medium or a computer network wherein program instructions are sent over optical or electronic communication links. Several inventive embodiments of the present invention are described below.

A system and method are disclosed for providing security for a computer network. Content sets are generated for a computer associated with the network. It is

10   determined whether a user should be routed to the generated content sets. If it is determined that the user should be routed to the generated content sets, a generated content set is selected and the user is so routed. Various actions and events may be recorded in a logfile, and the logfile is analyzed using regular expressions.

These and other features and advantages of the present invention will be presented

15   in more detail in the following detailed description and the accompanying figures, which illustrate by way of example the principles of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be readily understood by the following detailed description in conjunction with the accompanying drawings, wherein like reference numerals designate like structural elements, and in which:

5   Figure 1 is a block diagram of a general purpose computer system 100 suitable for carrying out the processing in accordance with one embodiment of the present invention.

Figure 2 is a schematic diagram of a system used in one embodiment to provide computer security.

Figure 3 is a flow chart illustrating a process used in one embodiment to provide

10   computer security using a trap system such as trap system 210 of Figure 2.

Figure 4 is a flowchart illustrating a process used in one embodiment to install a trap system, as in step 302 of Figure 3.

Figure 5 is an exemplary administration console display 500 used in one embodiment to provide a graphical user interface on the administration console for

15   configuration and control of the trap system.

Figure 6 is a flowchart illustrating a process used in one embodiment to generate file content for the trap, as required, e.g., in step 304 of Figure 3.

Figure 7 is a flowchart illustrating a process used in one embodiment to set the trap, as in step 306 of Figure 3.

Figure 8 is an illustration of a deception login screen 800 used in one embodiment to prompt an intruder who has been routed into the cage directory of the trap system to enter a login name.

Figure 9 is a flowchart illustrating a process used in one embodiment to keep an intruder in the trap, as in step 312 of Figure 3.

Figure 10 is a flowchart illustrating a process used in one embodiment to determine whether access to a particular file requested by an intruder is permitted, as in step 906 of Figure 9.

Figure 11A is a flowchart illustrating a process used in one embodiment to monitor the activity of an intruder, as in step 314 of Figure 3.

Figure 11B is a flow chart illustrating a process used in one embodiment to regenerate a virtual cage environment by using a product serial number as the seed for a pseudo random number generator.

Figure 11C is a flow chart illustrating a process used in one embodiment to hide the connection between the administrative console and the trap host system by using a "connectionless" port, as discussed above in connection with step 1104 of Figure 11A.

Figure 12 is a schematic diagram of a system used in one embodiment to provide such a test environment.

Figure 13 is a flowchart illustrating a process used in one embodiment to provide a virtual test environment to test the effect of a configuration change prior to implementing the configuration change on the actual computer system.

Figure 14 is a schematic diagram of a system used in one embodiment to provide computer security with multiple cages.

Figure 15 is a schematic diagram of a trap host system used in one embodiment to provide multiple cages.

Figure 16 is a flowchart illustrating a process used in one embodiment to provide computer security using a trap system such as trap system 1410 of Figure 14.

Figure 17 is a flowchart illustrating a process used in one embodiment to install a trap system, as in step 1602 of Figure 16.

Figure 18 is a flowchart illustrating a process used in one embodiment to set the trap, as in step 1606 of Figure 16.

Figure 19 is a flowchart illustrating a process used in one embodiment to handle a call to *kill* from within a cage.

Figure 20 is a flowchart illustrating a process used in one embodiment to handle a call to *bind* from within a cage.

Figure 21 is a flowchart illustrating a process used in one embodiment to handle a call to *listen* from within a cage.

Figure 22 is a flowchart illustrating a process used in one embodiment to handle a call to *connect* from within a cage.

Figure 23 is a flowchart illustrating a process used in one embodiment to handle a call to *getsockname* from within a cage.

5 Figure 24 is a flowchart illustrating a process used in one embodiment to handle a call to *ioctl* from within a cage.

Figure 25 is a schematic illustrating the configuration of *netstat* in one embodiment.

Figure 26 illustrates a regular expression query used in one embodiment to detect
10 a possible *sgid* exploit.

Figure 27 illustrates a regular expression query used in one embodiment to detect a possible *suid* exploit.

Figure 28 illustrates a regular expression query used in one embodiment to identify all processes.

15 Figure 29 illustrates a regular expression query used in one embodiment to identify certain processes.

Figure 30 illustrates a regular expression query used in one embodiment to identify shell-spawned processes.

Figure 31 illustrates a regular expression query used in one embodiment to identify incoming connections.

Figure 32 illustrates a regular expression query used in one embodiment to process keystrokes.

5   Figure 33 illustrates a regular expression query used in one embodiment to process screen output.

Figure 34 illustrates a regular expression query used in one embodiment to track monitored files.

DETAILED DESCRIPTION

A detailed description of a preferred embodiment of the invention is provided

below. While the invention is described in conjunction with that preferred embodiment,

it should be understood that the invention is not limited to any one embodiment. On the

5   contrary, the scope of the invention is limited only by the appended claims and the

invention encompasses numerous alternatives, modifications and equivalents. For the

purpose of example, numerous specific details are set forth in the following description in

order to provide a thorough understanding of the present invention. The present

invention may be practiced according to the claims without some or all of these specific

10   details. For the purpose of clarity, technical material that is known in the technical fields

related to the invention has not been described in detail so that the present invention is

not unnecessarily obscured.

Figure 1 is a block diagram of a general purpose computer system 100 suitable for

carrying out the processing in accordance with one embodiment of the present invention.

15   Figure 1 illustrates one embodiment of a general purpose computer system. Other

computer system architectures and configurations can be used for carrying out the

processing of the present invention. Computer system 100, made up of various

subsystems described below, includes at least one microprocessor subsystem (also

referred to as a central processing unit, or CPU) 102. That is, CPU 102 can be

20   implemented by a single-chip processor or by multiple processors. CPU 102 is a general

purpose digital processor which controls the operation of the computer system 100.

Using instructions retrieved from memory 110, the CPU 102 controls the reception and manipulation of input data, and the output and display of data on output devices.

CPU 102 is coupled bi-directionally with memory 110 which can include a first primary storage, typically a random access memory (RAM), and a second primary storage area, typically a read-only memory (ROM). As is well known in the art, primary storage can be used as a general storage area and as scratch-pad memory, and can also be used to store input data and processed data. It can also store programming instructions and data, in the form of data objects and text objects, in addition to other data and instructions for processes operating on CPU 102. Also as well known in the art, primary storage typically includes basic operating instructions, program code, data and objects used by the CPU 102 to perform its functions. Primary storage devices 110 may include any suitable computer-readable storage media, described below, depending on whether, for example, data access needs to be bi-directional or uni-directional. CPU 102 can also directly and very rapidly retrieve and store frequently needed data in a cache memory (not shown).

A removable mass storage device 112 provides additional data storage capacity for the computer system 100, and is coupled either bi-directionally or uni-directionally to CPU 102. For example, a specific removable mass storage device commonly known as a CD-ROM typically passes data uni-directionally to the CPU 102, whereas a floppy disk can pass data bi-directionally to the CPU 102. Storage 112 may also include computer-readable media such as magnetic tape, flash memory, signals embodied on a carrier wave, PC-CARDS, portable mass storage devices, holographic storage devices, and other

storage devices. A fixed mass storage 120 can also provide additional data storage

capacity. The most common example of mass storage 120 is a hard disk drive. Mass

storage 112, 120 generally store additional programming instructions, data, and the like

that typically are not in active use by the CPU 102. It will be appreciated that the

5   information retained within mass storage 112, 120 may be incorporated, if needed, in

standard fashion as part of primary storage 110 (e.g. RAM) as virtual memory.

In addition to providing CPU 102 access to storage subsystems, bus 114 can be

used to provide access other subsystems and devices as well. In the described

embodiment, these can include a display monitor 118, a network interface 116, a

10   keyboard 104, and a pointing device 106, as well as an auxiliary input/output device

interface, a sound card, speakers, and other subsystems as needed. The pointing device

106 may be a mouse, stylus, track ball, or tablet, and is useful for interacting with a

graphical user interface.

The network interface 116 allows CPU 102 to be coupled to another computer,

15   computer network, or telecommunications network using a network connection as shown.

Through the network interface 116, it is contemplated that the CPU 102 might receive

information, *e.g.*, data objects or program instructions, from another network, or might

output information to another network in the course of performing the above-described

method steps. Information, often represented as a sequence of instructions to be executed

20   on a CPU, may be received from and outputted to another network, for example, in the

form of a computer data signal embodied in a carrier wave. An interface card or similar

device and appropriate software implemented by CPU 102 can be used to connect the

APPLICATION FOR UNITED STATES PATENT

SYSTEM AND METHOD FOR COMPUTER SECURITY USING MULTIPLE CAGES

By Inventors:

Stephen Sorkin
810 Coleman Ave #16
Menlo Park, CA 94025
A Citizen of the United States

Michael Lyle
2844 Buena Knoll Court
San Jose, CA 95121
A Citizen of the United States

Robert F. Ross
151 Calderon #334
Mountain View, CA 94041
A citizen of the United States

James R. Maricondo
872 Ames Court
Palo Alto, CA 94303
A Citizen of the United States

Assignee: Recourse Technologies, Inc.

computer system 100 to an external network and transfer data according to standard protocols. That is, method embodiments of the present invention may execute solely upon CPU 102, or may be performed across a network such as the Internet, intranet networks, or local area networks, in conjunction with a remote CPU that shares a portion

5   of the processing. Additional mass storage devices (not shown) may also be connected to CPU 102 through network interface 116.

An auxiliary I/O device interface (not shown) can be used in conjunction with computer system 100. The auxiliary I/O device interface can include general and customized interfaces that allow the CPU 102 to send and, more typically, receive data

10   from other devices such as microphones, touch-sensitive displays, transducer card readers, tape readers, voice or handwriting recognizers, biometrics readers, cameras, portable mass storage devices, and other computers.

In addition, embodiments of the present invention further relate to computer storage products with a computer readable medium that contain program code for

15   performing various computer-implemented operations. The computer-readable medium is any data storage device that can store data which can thereafter be read by a computer system. The media and program code may be those specially designed and constructed for the purposes of the present invention, or they may be of the kind well known to those of ordinary skill in the computer software arts. Examples of computer-readable media

20   include, but are not limited to, all the media mentioned above: magnetic media such as hard disks, floppy disks, and magnetic tape; optical media such as CD-ROM disks; magneto-optical media such as floptical disks; and specially configured hardware devices

such as application-specific integrated circuits (ASICs), programmable logic devices (PLDs), and ROM and RAM devices. The computer-readable medium can also be distributed as a data signal embodied in a carrier wave over a network of coupled computer systems so that the computer-readable code is stored and executed in a

5   distributed fashion. Examples of program code include both machine code, as produced, for example, by a compiler, or files containing higher level code that may be executed using an interpreter.

The computer system shown in Fig. 1 is but an example of a computer system suitable for use with the invention. Other computer systems suitable for use with the

10   invention may include additional or fewer subsystems. In addition, bus 114 is illustrative of any interconnection scheme serving to link the subsystems. Other computer architectures having different configurations of subsystems may also be utilized.

Figure 2 is a schematic diagram of a system used in one embodiment to provide computer security. The system includes a computer network 202 to which the operator of

15   the computer network wishes to limit access to authorized users. Computer network 202 is comprised of a plurality of network devices 204. The plurality of network devices 204 may include, for example, individual computer work stations, network servers, printers, and any number of other devices such as may be found in a typical computer network, such as a local area network (LAN) or wide area network (WAN). Computer network

20   202 also includes a Internet access server 206 configured to enable users of host computer systems connected to the computer network 202 to access the Internet and in particular to access web pages via the World Wide Web by sending and receiving

hypertext transfer protocol (HTTP) transmissions. Computer network 202 also includes a firewall 208 interposed between Internet access server 206 and the network connection to the Internet. Firewall 208 may be either a firewall, or a router with firewall functionality, configured to route authorized users to Internet access server 206 and to detect and route

5 unauthorized users to the trap system described below.

The system shown in Figure 2 also includes a trap system 210. Trap system 210 is comprised of a trap host system 212 in which a virtual cage 214 is established, as described below. Trap system 210 also includes an administration console 216 connected to trap host system 212 and configured to enable a system administrator (or other

10 authorized user) to control the configuration of trap host system 212 and virtual cage 214. Trap system 210 also includes a database 218 used to store data relating to activities within trap host system 212 and virtual cage 214.

The system shown in Figure 2 is designed to protect the computer network 202 from being accessed or otherwise compromised by an intruder who is attempting to gain

15 access to computer network 202 via the Internet. Figure 2 shows an exemplary intruder's system 220 such as might be used by a would-be intruder to attempt to gain access to the computer network 202 via the Internet.

Figure 3 is a flow chart illustrating a process used in one embodiment to provide computer security using a trap system such as trap system 210 of Figure 2. The process

20 begins with step 302 in which a trap system such as trap system 210 of Figure 2 is installed. In step 304, the file content for a deception environment to be presented to

would-be intruders is created. Examples of the content of the deception environment include fictitious content generated automatically as described below; non-confidential (i.e., public) files drawn from the computer network being protected, such as computer network 202 of Figure 2; or a combination of fictitious and non-confidential file content.

5   In step 306, a trap is established within the trap system. For example, a virtual cage such as virtual cage 214, shown in Figure 2 may be established within a trap host system, such as trap host system 212 of Figure 2, by establishing a file directory for the cage and copying the operating system of the trap host system -- but not the modifications and additions to the operating system described below that function to

10   monitor the intruder's actions, keep the intruder in the cage, and prevent the intruder from detecting that the intruder is in the cage -- and the file system of the trap host system into the directory.

     In step 308, a would-be intruder is detected, as described more fully below. In step 310, the would-be intruder is routed into the trap system, such as trap system 210 of

15   Figure 2, as described more fully below. Once the intruder has been routed into the trap, in step 312 affirmative efforts can be made to ensure that the intruder does not break out of the trap system and gain access to the portions of computer network 202 that are being protected from unauthorized access. In step 314, the activity of the intruder within the trap system is monitored, as described more fully below.

20   Once the activity of the intruder has ceased, either because the intruder has discontinued the attempt to access computer network 202 or because the system

administrator has terminated the intruder's connection with the system, it is determined in step 316 whether the changes to the configuration to the trap system that were made by the intruder during the attack will be kept in place. For example, a system administrator might wish to leave changes made by an intruder in place if the system administrator

5   believes the same intruder may attempt a future attack and might realize that he or she has been routed into a deception environment, as opposed to gaining access to the true computer network, if the changes made by the intruder in the prior attack were not still present. If it is determined in step 316 that the changes will be kept, the process shown in Figure 3 ends and the trap remains in place, as modified by the intruder, unless or until a

10   future intruder is routed into the trap or the trap is reset. If it is determined in step 316 that the changes made by a particular intruder will not be kept, the process proceeds to step 318 in which the trap is reset to eliminate the changes made by the intruder. In one embodiment, the trap is reset by regenerating the trap to restore the trap environment to the condition it was in at the time the intruder was first routed into the trap. In one

15   embodiment, additional content is added when the trap is regenerated to make it appear that additional content was created by users of the computer network during the time period from the last update of the trap to the time the trap was reset.

Figure 4 is a flowchart illustrating a process used in one embodiment to install a trap system, as in step 302 of Figure 3. The process begins with step 402 in which a trap

20   host system is installed. In one embodiment, the trap host system is a computer, such as an Intel or SPARC computer, running a Unix operating system in the form of a Solaris 2.x operating system (such as Solaris 7). In one embodiment, application programs that

the user of the trap system wishes to have appear in the deception environment may be installed in the trap host system prior to the installation of the trap system software and the establishment of the virtual cage environment into which the operating system and file system of the trap host system will be copied. In one embodiment, probabilistic data

5   combined with random number data from a pseudo random number generator are used to determine which application programs will appear in the deception environment. In one embodiment, the nature of the business or other organization that uses the computer network influences which application programs are selected. For example, a financial institution may have different application programs, and different types of files, than a

10   law firm.

In step 404, an administration console, such as administration console 216 of Figure 2, is installed. The administration console is a second computer system connected to the trap host system. The administration console is used to configure and control the operation of the trap host system. In addition, the administration console receives

15   logging information from the trap host system concerning the activities of the intruder within the trap host system. In one embodiment, administration console 216 is a computer system running either a UNIX or a Windows operating system. The administration console uses its connection to the trap host system to retrieve log and configuration information for the purpose of displaying the information to the system

20   administrator.

In step 406, the trap host system is configured. As noted above, the administration console 216 is used to select configuration options for the trap software,

once the trap software has been installed in the trap host system. In one embodiment, upon installation, the trap software automatically configures the trap host system in accordance with the preferences selected by the system administrator or other authorized user of the system by means of the administration console and randomly generated

5   variations in certain system settings, as described more fully below.

The process shown in Figure 4 continues with step 408 in which a network connection is made between the trap system and the router or firewall used in the computer network being protected to detect and route would-be intruders into the trap system. In one embodiment, network connections are made between the trap host system

10   and the router or firewall for all or selected ones of the remote access services that an intruder might use to attempt to gain unauthorized access to, or control over, a target computer or computer network. In one embodiment, the trap host system operating system is the Solaris 7 operating system and the remote access services for which a network connection is established include FTP (file transfer protocol), telnet, and/or other

15   services considered to be in the so-called "demilitarized zone", or "DMZ", of the network being protected.

In step 410, the policy editor of the router or firewall, which is typically provided as part of the software associated with a router or firewall, is used to establish policies which will route likely intruders to the trap host system. Such policies may include,

20   where supported by the particular router or firewall being used, a policy that attempts to gain access to the computer network via a port or service not normally used by the computer network, but known to be exploited by hackers and other intruders to gain

access to computer networks, such as the FTP and telnet ports, for example, can be routed to the corresponding port of the trap host system. In one embodiment, a would-be intruder is permitted to see the devices behind the router or firewall. If the would-be intruder seeks to gain access to the virtual cage environment, which can be configured to

5   appear to be an interesting and easy target for intrusion (e.g. because services that are known to be exploitable to gain unauthorized access or control, such as FTP and telnet, will be available), the router or firewall can be configured in step 410 to route the intruder to the appropriate port of the trap host system using well known network address translation (NAT) techniques. In one embodiment, a would-be intruder cannot see the

10   devices behind the router or firewall and any attempt to access a prohibited service on any network system is routed instead to the trap host system using NAT.

Figure 5 is an exemplary administration console display 500 used in one embodiment to provide a graphical user interface on the administration console for configuration and control of the trap system. The administration console display 500

15   includes a menu display area 502 in which menu choices are displayed. As shown in Figure 5, in one embodiment, the major headings "General", "Decoy User Names", "Logging", "Alerting", and "Advanced" are displayed in menu display area 502. In one embodiment, selection of a major menu listing results in the subheadings under that menu listing being displayed. Display 500 also includes an instruction display area 504 in

20   which instructions relating to the current menu selection are displayed. Display 500 also includes an input area 506 in which the system administrator or other user either enters

data or selects an option from a pick list to provide input with respect to the current menu selection.

In one embodiment, the "General" menu provides options for entering the name of the company using the trap system; entering a license key or serial number for the system; entering a host name to be used in the contents created for the deception environment to identify the host associated with certain content; and to designate a domain name to be used for similar purposes, such as to be included as the domain name for Internet e-mail addresses for the fictitious and other user names used in the e-mail messages generated to be included in the deception environment. In one embodiment, the menu selection "Decoy User Name" enables the system administrator to provide the full name and a login or user name for from one to five individuals. Such an option may be used to provide the name of from one to five prominent and publicly-known individuals associated with the computer system being protected, such as the chief executive officer and/or president of the company that uses the system.

In one embodiment, the menu option labeled "Logging" includes options that enable the system administrator to route logging information from the trap system to a remote logging device, such as by providing the DNS name or IP address of the remote logging server. In addition, the "Logging" menu in one embodiment includes an option to either enable remote logging, as described above, or to disable remote logging and to have the log information spooled only to the trap host system. Finally, the "Logging" menu option in one embodiment includes an option that permits the system administrator

to designate the name of the network interface used to gather information on an intruder's network activity, for example for use in later tracing the source of an intruder's attack.

In one embodiment the menu heading "Alerting" provides options for controlling the manner in which alerts regarding intruder activity is provided and the criteria used to
5　determine when such an alert should be sent. The purpose of such an alert is to advise the system administrator that an intruder has gained a certain level of access to or control over the trap system. Providing such an alert enables the system administrator to more closely monitor the intruder and, if necessary, to cut off the intruder's connection to the system. The degree to which an intruder has gained unauthorized access or control is
10　sometimes referred to as the extent to which the security of the system or network has been compromised by the intruder. In one embodiment, the options under the menu heading "Alerting" include the options to designate an e-mail address to be used to provide alerts, a fictitious subject line to be used in such e-mail messages, and an option for selecting an alert threshold.

15　For example, in one embodiment, one of five alert thresholds may be selected. The lowest threshold provides that no e-mail alert messages will be sent regardless of the type or severity of the compromise achieved by the intruder. A somewhat higher threshold provides for an e-mail alert message to be sent if the trap host computer system experiences a fatal error, for example if the host runs out of disk space. The next higher
20　level provides for an e-mail alert message to be sent in a clear case of compromise such as if a new process has started within the virtual cage environment in the trap host system. The next somewhat higher level of alert provides for an e-mail alert message to

be sent in situations that indicate a possible security compromise, such as if multiple port connections are opened by an intruder in an attempt to determine which processes are currently running on the host system. The most sensitive and final level of alert provides for an e-mail alert message to be sent whenever the virtual cage environment experiences

5   any traffic, regardless of type. At this heightened level, alert messages may be generated based on intruder activity within the cage environment even in cases where there is no information indicating that the cage has been compromised or is in risk of being compromised.

Finally, the menu heading "Advanced" in one embodiment provides options for

10   customizing the file content for the virtual cage environment and for making more complex configuration changes, to accomplish such goals as optimizing system performance or to otherwise tailor the trap system to the specific needs of a particular user.

Referring further to Figure 5, the administration console display 500 also includes

15   a back button 508 and a next button 510 used to navigate back to the previous menu option or forward to the next menu option, respectively. The display 500 also includes a revert button 512 used to cancel a configuration change entered at the administration console and revert to the configuration settings that were in place prior to any changes being made. Display 500 also includes an update button 514 used to update a file

20   maintained locally at the administration console to store configuration changes entered at the administration console but not yet applied to the trap host system. Display 500 also includes an apply button 516 used to apply configuration changes entered at the

administration console to the trap host system. Finally, display 500 includes a reboot button 518, which causes the trap host system to reboot. In one embodiment, it is necessary to reboot the trap host system in order for configuration changes to be implemented in the trap host system.

5   Figure 6 is a flowchart illustrating a process used in one embodiment to generate file content for the trap, as required, e.g., in step 304 of Figure 3. The process begins with step 602 in which operating system settings are generated automatically for the operating system installed in the trap host system. Operating system settings are generated automatically, with random variations included, to avoid having the same

10   operating system configuration for each trap system. If such variations were not introduced, would-be intruders might be able to recognize that a system is a trap system provided by a particular manufacturer based on the presence of a standard operating system configuration used by the manufacturer.

       Next, in step 604, information is generated automatically concerning the hardware

15   installed on the trap host system, the configuration of such hardware, and other information concerning the configuration of the trap host system.

       The process continues with step 606 in which selected real data and files are received and loaded. Any selected real files to be made available in the trap system, such as publicly-available documents or information, are stored in the file system or the trap

20   host system. Real data to be used to fill in document templates, such as the names of key employees or other publicly-known individuals, are stored in the applicable database.

Then, in step 608, a database of fictitious names to be used in automatically generated e-mail and other documents is generated. A unique key or serial number provided with each copy of the software for the trap system serves in one embodiment as the seed for a pseudo random number generator. Numbers from the pseudo random

5 number generator are used in conjunction with probabilistic data concerning the occurrence of first and last names from a database of names to generate a list of fictitious user names to be used to generate file content for a particular trap system.

The process continues with step 610 in which fictitious file content, such as fictitious e-mail, word processing document, spreadsheet, and other file content, is

10 generated. In one embodiment, e-mail and other document templates are provided which require data values such as dates, names, product names, and other types of information to be inserted. Random numbers from a pseudo random number generator and probabilistic data are used to select a set of file templates to be used for the file content of a particular trap system. The set of templates to be used for any given system will be

15 unique because the pseudo random number generator uses the unique product serial number or key for each particular system as the seed for the pseudo random number generator. Once the set of templates has been selected, the data values for each of the inputs required by each template are provided by using the pseudo random number generator and probabilistic data to select values from various databases of possible values

20 provided for each type of input required by the templates.

An exemplary e-mail template used in one embodiment for generating an e-mail message announcing a meeting for a project identified by a code name follows:

&MEETING: 10
To: @EMPLOYEE
Subject: Meeting re @PROJECT

5   The meeting re @PROJECT will take place on @DAY, @MONTH @1TO28, at @TIME. The meeting will be held in @NAME=1's office. Coffee and rolls will be served. Please RSVP to @NAME=2 NLT (@DAY-1).

In the above exemplary template, the entry "&MEETING: 10" indicates that the

10   template is a meeting announcement template with a relative probability of occurrence of

10. The relative probability of occurrence is a weight value for the template, which is

based on studies of actual file systems found in a typical network server. The sum of all

of the relative probability values for all templates appears at the top of the template file,

and the relative likelihood that the above particular template will be selected at random

15   from among the entire body of templates is determined by dividing the weight for the

template, 10, by the sum of all of the weights. For example, if the sum of all of the

weights were 1,000, the probability of the above template being selected would be

10/1,000. By comparison, a product launch announcement template might have a weight

of only 1. The probability of such a template being selected would be 1/1,000, or about

20   one tenth that of the above template. This would indicate that a product launch

announcement e-mail would be one tenth as likely as a meeting announcement e-mail to

be found in a typical network server. As described above, in one embodiment the

selection of a set of templates for the initial file content for the trap file system would be

based on the probability weight values and numbers generated by a pseudo random

25   number generator.

The values of the variables @EMPLOYEE, @PROJECT, @DAY, @MONTH, @1TO28, @TIME, @NAME=1, and @NAME=2 in the above exemplary template are selected in one embodiment from corresponding files comprising possible values and a corresponding probability weight for each possible value. A number generated by a

5   pseudo random number generator is used, in combination with the probability weights, to select the specific value for a particular instance. For example, the value of the variable @EMPLOYEE is selected at random from a file comprising names of fictitious employees and associated data, such as network usernames, e-mail addresses, and host system identification information. In one embodiment, the variable @EMPLOYEE is

10   replaced with the e-mail address of from one to ten fictitious employees (and other information required for a file comprising an e-mail to the employee(s)), with the precise number of recipients being determined at random. In a similar manner, a day of the week would be selected as the value of the variable @DAY, a month for the variable @MONTH, a number from 1 to 28 for the variable @1TO28, and a time (e.g., at half

15   hour increments during business hours) for the variable @TIME, would be chosen at random from corresponding files of possible values.

A similar technique may be used to select values for the variables @NAME=1 and @NAME=2 from a file containing the fictitious user names, created as described above. The annotations "=1" and "=2" indicate that a different name should be selected

20   for each variable.

For certain types of variables, probabilities of occurrence would be considered in one embodiment in selecting the value. For example, the value for the variable @PROJECT is selected in one embodiment from a file such as the following:

```
        @PROJECT: 90
5       10:  SPEAR
        20:  WIN
        20:  SPEED
        10:  NORMANDY
        10:  STORM
10      20:  VICTORY
```

In the above file, the entry "@PROJECT: 90" identifies the files as containing possible values for the variable @PROJECT and indicates the sum of the probability weights for the possible values is 90. (In one embodiment, if the relative probability of occurrence of each value were the same, the number after the colon would be the total number of possible values in the file and the relative weight of each value would be assumed to be 1.) Each of the remaining entries in the file comprises a probability weight followed by a possible value. For example, the entry "10: SPEAR" has a probability weight of 10 and a value of "SPEAR". The weight indicates the value SPEAR has a 10 in 90 (i.e., one in nine) probability of occurrence. The value chosen for a particular instance of a template is selected using a number generated by a pseudo random number generator and the probabilistic data.

In one embodiment, spelling, grammatical, and typographical errors are introduced into at least certain portions of the generated file content. Probabilistic data concerning the occurrence of such errors and a pseudo random number generator are used to determine the nature and location of the errors that are introduced.

In one embodiment, additional file content is generated, in the manner described above, at random intervals after the initial set of file content has been generated. In one embodiment, a pseudo random number generator is used to determine the intervals at which additional file content is generated. In one embodiment, file content is generated

5   at more frequent intervals during certain times of the day, such as business hours, than during other times of the day. Additional file content is generated over time in order to provide a more realistic deception environment. For example, if an intruder accesses the trap system on one occasion and later returns to access the trap system in the future, the intruder may become suspicious if no additional file content has been generated in the file

10   system since the initial attack. In addition, even if an intruder only accesses the file system on one occasion, the intruder may become suspicious if the system has been installed for a considerable period of time and no additional file content has been generated since the time of installation.

Figure 7 is a flowchart illustrating a process used in one embodiment to set the

15   trap, as in step 306 of Figure 3. The process begins with step 702 in which a cage is established within the trap host system. In one embodiment, this is accomplished by creating within the file system of the trap host system a new directory to contain the file structure for the cage.

In step 704, the operating system of the trap host system is copied into the cage

20   directory. As described more fully below, the interface to the operating system kernel is modified to monitor the intruder's actions (e.g., by generating log data regarding an intruders activities), keep the intruder in the cage, and prevent the intruder from detecting

that the intruder is in the cage. The files and programs that perform these latter functions are not copied into the cage. In step 706, the file system of the trap host system is copied into the cage directory.

By copying the operating system of the trap host system and the file system of the trap host system into the cage directory, it becomes easier to route an intruder into the cage directory and present to the intruder a deception environment that leads the intruder to believe that the intruder has successfully gained access to the operating system and file system of the computer the intruder is targeting. From time to time, additional file content is generated and added to the copy of the file system in the cage directory, as described above, to provide a more realistic deception environment.

Once an intruder has been detected and routed into the cage directory of the trap host system, a deception environment is presented to the intruder. The intruder interacts with the instance of the operating system running in the virtual cage environment. Figure 8 is an illustration of a deception login screen 800 used in one embodiment to prompt an intruder who has been routed into the cage directory of the trap system to enter a login name. In one embodiment, the trap host system is configured to make it relatively easy for an intruder to obtain a login or user name and the corresponding password that will enable the intruder to gain access to the trap system using well-known hacking techniques.

Figure 9 is a flowchart illustrating a process used in one embodiment to keep an intruder in the trap, as in step 312 of Figure 3. The process begins with step 902 in which

a request to access a file within the cage directory is received from the intruder. In one

embodiment, a software module is provided to serve as a filter between requests made by

an intruder to access a file, on the one hand, and the copy of the file system contained in

the cage directory of the trap system, on the other hand. Such filtering software is used to

5   prevent the intruder from accessing files that might enable the intruder to discover that

the intruder is in a trap system, and not an actual system, as described more fully below.

In step 904, the filtering software sends log information to the user-specified

destination for logging data concerning activities of intruders.

The process continues with step 906 in which it is determined whether the

10   intruder is permitted to access the particular file the intruder has requested. In one

embodiment, the filtering software referred to above, and described more fully below,

makes this determination. If it is determined in step 906 that the intruder is not permitted

to access the requested file, the process proceeds to step 908 in which an indication is

provided to the intruder that the requested file does not exist. If it is determined in step

15   906 that the intruder is authorized to access the requested file, the process proceeds to

step 910 in which the intruder is provided access to the copy of the requested file

contained within the cage directory in the trap system.

Figure 10 is a flowchart illustrating a process used in one embodiment to

determine whether access to a particular file requested by an intruder is permitted, as in

20   step 906 of Figure 9. The process begins at step 1002 in which it is determined whether

the intruder is attempting to request a file that is at a level within the trap host system file

structure that is above the highest level of the cage file structure, i.e., above the directory

created to hold the file structure and operating system for the cage. For example, in one

embodiment, the trap host system operating system is Solaris 7 ™. In the Solaris 7

operating system, the command "/../proc", for example may be used to gain access to the

5   directory level above the file "proc", which would normally be in the highest level of the

file structure for a system such as the trap host system. If an intruder were able to use

this command to move above the "proc" file in the cage directory (which is a copy of the

proc file of the trap host system copied into the cage directory), the intruder likely would

realize that the intruder has been contained within the cage directory and, once the

10   intruder has broken out of the cage directory, the intruder is much more likely to be able

to compromise the trap host system. In one embodiment, the "/../proc" command or

similar commands that might be used to access a level of the trap host system file

structure that is above the highest level of the cage file structure are filtered by a software

module which recognizes such commands, prevents them from being executed, and

15   provides an indication (as in step 1002) that an attempt is being made to move above the

highest level of the cage file structure.

If it is determined in step 1002 that an attempt is being made to move above the

highest level of the cage file structure, the process proceeds to step 1004 in which access

to the requested file structure level is denied and an indication is provided to the intruder

20   that the requested file does not exist, in accordance with step 908 of Figure 9. If it is

determined in step 1002 that an attempt is not being made to move above the highest

level of the cage file structure, the process proceeds to step 1006 in which it is

determined whether the intruder is making an attempt to access a blocked network data file. For example, in the Solaris 7 operating system, all network devices have a major and minor number associated with them. It is known in the art of computer security and the art of computer hacking that files associated with certain device numbers are

5   susceptible to being used to gain unauthorized access to or control over a target computer system. For example, in one embodiment the trap host system uses the Solaris 7 operating system for which the device files for devices that have a major number 7 and a minor number in the range of 0-7, or devices that have a major number 11 and a minor number 7, may be exploited by an intruder to gain an unauthorized level of access to or

10   control over a target computer system. As a result, in one embodiment, it is determined in step 1006 whether the intruder is attempting to access the device files associated with a device having a major and minor number in one of the ranges listed above.

If it is determined in step 1006 that an attempt is being made to access a blocked network data file, the process proceeds to step 1008 in which access to the requested file

15   is denied, and an indication is provided that the file does not exist in accordance with step 908 of Figure 9. If it is determined in step 1006 that an attempt to access a blocked network data file is not being made, the process proceeds to step 1010 in which it is determined whether an attempt is being made to access a process file for a process running outside of the virtual cage environment. Each computer operating system

20   provides a way to monitor the processes or tasks currently being performed by the host system. In the Solaris 7 operating system, for example, a process table is provided in a file contained within the operating system's virtual file system. The process table is

accessed by entering a file name in the directory "/proc". In one embodiment, a software

module is used to filter access to the "proc" file to limit an intruder's access to files

associated with processes running within the cage environment and to prevent access to

processes running on the trap host system outside of the virtual cage.

5   If it is determined in step 1010 that an attempt is being made to access a process

file for a process running outside of the cage environment, the process of Figure 10

proceeds to step 1012 in which access to the requested file is denied, and an indication is

provided that the file does not exist in accordance with step 908 of Figure 9. If it is

determined in step 1010 that an attempt is not being made to access a process file for a

10   process running outside of the cage environment, the process proceeds to step 1014 in

which access to the requested file is permitted in accordance with step 910 of Figure 9.

In one embodiment, at least one of the steps of the process illustrated in Figure 10

is implemented by replacing one or more operating system functions in the system entry

(or "sysent") table with a new program designed to perform the above-described filtering

15   function. In one embodiment, the new program returns the output of the original

operating system function if access to a requested file (or process) is permitted (i.e., the

file or process is within the virtual cage) and returns an indication that the file (or

process) does not exist, if the file (or process) is not inside the cage. In one embodiment,

a similar approach is used to modify the function that responds to system calls such as

20   "kill", in order to permit intruders to terminate only processes running inside the cage.

Figure 11A is a flowchart illustrating a process used in one embodiment to monitor the activity of an intruder, as in step 314 of Figure 3. The process begins at step 1102 in which a log of the intruder's actions is maintained. In one embodiment, the software modules used to filter requests to access various types of files send information

5   concerning each request by the intruder to access a file to a log file used to store information concerning the files requested by an intruder. In one embodiment, the trap system can be configured to log either each command entered by an intruder or to log each keystroke entered by the intruder. In addition to information concerning the intruder's actions sent by the filtering software modules described above, information

10   concerning the processes running within the virtual cage environment and what specific tasks each process is performing is available from the existing process file system (/proc) and is logged along with the log information derived from the filtering software modules.

As noted above, the intruder is prevented from becoming aware of the monitoring and logging processes by operation of the software module that filters the intruder's

15   requests to access files within the process file system to prevent access to files relating to the monitoring and logging processes.

The process shown in Figure 11A also includes a step 1104 in which log information is made available to the system administrator or other user of the trap system at a graphical user interface (GUI) presented at a control station such as administration

20   console 216 of Figure 2. This enables a system administrator or other user of the trap system either to perform an analysis of an intruder's actions subsequent to an attack or to monitor the actions of an intruder in real time, so as to be in a position, for example, to

terminate the connection of the intruder to the trap host system if there is a risk the

intruder may gain access to files outside of the virtual cage environment. In one

embodiment, the connection of the administration console or other control system

providing a graphical user interface for the trap system is hidden from detection by an

5   intruder by use of a so-called "connectionless" port to provide for the exchange of

information between the administration console and the trap host system, as described

more fully below in connection with Figure 11C.

The process illustrated in Figure 11A also includes step 1106 in which it is

determined whether the alert conditions established at the time the trap system was

10   configured have been met. For example, in one embodiment, as described above, the

"normal" level of alert conditions provides for the trap system to send an alert e-mail in a

situation that indicates a possible security compromise, for example if multiple port

connections are open, which may indicate that an intruder is attempting to determine

which processes are currently running on the host system. As described above, a more

15   sensitive level of alert may be established in which an alert e-mail message would be sent

whenever the virtual cage environment experiences any activity, regardless of the type.

If it is determined in step 1106 that the alert conditions have not been met, the

process proceeds to step 1108 in which the monitoring and logging of the intruder's

activities continues until the intruder leaves the system. If it is determined in step 1106

20   that the alert conditions have been met, the process proceeds to step 1110 in which an

alert is sent to the system administrator (or other designated user). In one embodiment,

the alert is an e-mail message sent to the system administrator. In one embodiment, a

subject line provided as part of the system configuration process is used to identify the nature of the message to an authorized individual who sees the subject line. If an alert has been sent in step 1110, the process continues with step 1112 in which the monitoring and logging of the intruder's activities continues either until the intruder voluntarily

5 leaves the system or until the intruder's connection to the system is terminated by the system administrator, for example by regenerating the virtual cage environment, rebooting the trap host system, or changing the firewall rule set to no longer permit the intruder to access the trap host system.

The automatically logged information can be used to analyze the strategies and

10 techniques used by the intruder to gain access to and attempt to gain control of the system. In one embodiment, another approach used to evaluate the activities of an intruder once an intruder has exited the system is to make a copy of the file system of the virtual cage environment and then to regenerate the virtual cage environment, as described above, and compare the regenerated virtual cage environment, which will not

15 have any of the changes made by the intruder, with the copy of the virtual cage environment as modified by the activities of the intruder. The log file may be processed as will be described herein.

In one embodiment, a unique key is used to seed the pseudo random number generator used to generate content for the file system, as described above. In one

20 embodiment, the key is the serial number of the copy of the trap software provided for a particular installation. Using a unique key to seed the pseudo random number generator ensures that the content of each trap system installed will be unique. The use of the same

key to seed the pseudo random number generator each time the virtual cage environment for a particular installation is regenerated results in the same content being created each time the cage is regenerated. As a result, a returning intruder will see all of the same file content that was in the cage during the intruder's previous attack, even if the cage has

5   been regenerated. If the changes made by the intruder during a prior attack were kept (i.e., the cage was not regenerated), the intruder will see the effects of the intruder's previous attack in the virtual cage environment. If the cage has been regenerated since a prior attack, the file system will contain the same file content the intruder saw during the previous attack, but will not contain changes made or caused by the intruder's activities.

10   This is the same environment an intruder would expect to see if the system had been reconstructed, such as from back-up tapes. In either event, the intruder sees a sufficiently familiar environment that the intruder likely will continue to be deceived.

Figure 11B is a flow chart illustrating a process used in one embodiment to regenerate a virtual cage environment by using a product serial number as the seed for a

15   pseudo random number generator. The process begins with step 1120 in which a product serial number is received. In step 1122, the product serial number is used as the seed for a pseudo random number generator used to generate file content for the virtual cage environment, as described above. In step 1124, it is determined whether a command to regenerate the trap has been received. If a request to regenerate the trap has not been

20   received, the process ends. If a request to regenerate the trap has been received, the process returns to step 1122 in which the product serial number is used once again as the

seed for the pseudo random number generator used to generate file content for the virtual cage environment.

Figure 11C is a flow chart illustrating a process used in one embodiment to hide the connection between the administrative console and the trap host system by using a "connectionless" port, as discussed above in connection with step 1104 of Figure 11A.

A typical way to connect such an administration console to a system such as the trap host system would be to use a connection that employs transmission control protocol (TCP), in which many packets of information are assembled together to appear as a uniform stream of information exchanged between the administration console and the trap host system. The shortcoming of this approach in the context of a system such as the trap system described herein is that an intruder would be able to see a connection that uses TCP as a continuously live connection to the trap host system. An intruder may become suspicious if the intruder can see that such a live connection exists.

In one embodiment, this shortcoming is avoided by employing a user datagram protocol (UDP) connection to connect the administration console to the trap host system. Unlike a TCP connection, a UDP connection does not result in many packets of data being assembled and transmitted as a uniform stream of information. Instead, each packet of information is sent with a hashed message authentication code (HMAC) used to identify the packet as having originated from an authorized source. Each packet is accepted at the receiving end if the required HMAC is present in the packet. In one embodiment, if the required HMAC is not present in a packet, the administration console

replies with the Internet Control Message Protocol (ICMP) packet that would be sent if the port were not in use.

Unlike TCP, UDP does not require a communication channel to be established and maintained between the administration console and the trap host system in order for data to be exchanged between the two systems. When an authorized user logs into the administration console to view logging information, the user enters a password and the administration console generates a key that will be used to determine the HMAC that is required to be included in a valid transmission to the trap host system. Data packets sent by the trap host system to the administration console that contain the required HMAC will be accepted and acted on by the administration console system. If an intruder, on the other hand, sends a packet to the administration console via the UDP port in an attempt to determine if the trap host system is communicating with a device connected to the port (i.e., software is bound to the port), the administration console will see that the required HMAC is not present and will reply with the packet that would be sent if the port were not in use, as described above. As a result, the intruder will be led to believe that the port is not in use.

The process shown in Figure 11C begins with step 1140, in which a user name and password are received at the administration console. In step 1142, a key for the session is provided. In one embodiment, the key is randomly generated. In one embodiment, the key is derived from the password. In step 1144, a message is received at the administration console via the connection to the trap host system. In step 1146, it is determined whether the incoming message contains the required HMAC.

If it is determined in step 1146 that the incoming message does not contain the required HMAC, the process proceeds to step 1148 in which the ICMP packet that would be provided if the port of the trap host system to which the administration console is connected were not in use is sent in response to the incoming message. If it is determined in step 1146 that the incoming message does contain the required HMAC, the process continues with step 1150, in which the incoming message is accepted by the administration console and the administration console takes appropriate responsive action, for example by responding to a command or query from the trap host system.

In step 1152, it is determined whether the session has ended, for example by determining whether the user has logged out of the administration console. If it is determined in step 1152 that the session has ended, the process ends. If it is determined in step 1152 that the session has not ended, the process returns to step 1144 in which the next incoming message, if any, is received.

In addition to providing computer security, the system and methods described herein may also be used for other purposes. For example, in one embodiment the techniques described above are used to provide a test environment to test the impact of a configuration change on a computer system without placing the actual files and data stored on the computer system at risk. Figure 12 is a schematic diagram of a system used in one embodiment to provide such a test environment. The system 1200 includes a network server 1202 in which a virtual test environment 1204 is established in the same manner as the virtual cage environment described above. One or more network devices 1206 are connected to the network server 1202 by means of a network bus 1208. A

remote system 1210 is configured to connect to network server 1202 by means of the Internet. An administration console 1212 is connected to the network server 1202 to be used to configure the network server and test environment, and to monitor activities in the test environment, similar to the administration console in the above-described security

5   embodiment.

Figure 13 is a flowchart illustrating a process used in one embodiment to provide a virtual test environment to test the effect of a configuration change prior to implementing the configuration change on the actual computer system. The process begins with step 1302 in which the software for providing the virtual environment is

10   installed in the server or other computer system in which the configuration change is to be made. Next, in step 1304, a virtual test environment is established in the same manner as described above for establishing a cage environment in the trap host system in a security embodiment. Specifically, a test environment directory is established and the network server operating system and file system are copied into the virtual test

15   environment.

Then, in step 1306, the contemplated change in configuration of the network server is implemented only in the test environment. For example, the configuration change may be the installation of a new software application. Alternatively, the configuration change may be the installation of a new network device on the network

20   bus, or the connection of a new remote system via the Internet or some other means of remote access to the network server.

Next, in step 1308, the server is operated with the configuration change having been implemented in the test environment.

In step 1310, data concerning the operations of the server within the test environment is logged. In one embodiment, data concerning the processes running on the server, and in particular processes running within the virtual test environment, is provided by the operating system kernel and sent to the administration console for storage in the database.

In step 1312, logged data is analyzed to determine the effect of the configuration change on the virtual test environment. In one embodiment, a copy of the virtual test environment is made and then the virtual test environment is regenerated to restore the virtual test environment to the condition it was in before the configuration change was made. Then, the copy of the virtual test environment as modified by the configuration change is compared to the regenerated virtual test environment to analyze all of the effects of the configuration change.

The process continues with step 1314 in which it is determined whether the configuration change created any problems in the configuration or operation of the server within the virtual test environment. If the configuration change did create a problem, the process proceeds to step 1316 in which the configuration change is reversed and the server is restored to the condition it was in prior to the configuration change. If it is determined in step 1314 that the configuration change did not result in any problem in the virtual test environment, the process proceeds to step 1318, in which the configuration

change is implemented in the server outside of the virtual test environment and the server is operated normally with the configuration change implemented.

Figure 14 is a schematic diagram of a system used in one embodiment to provide computer security. Similar to Figure 2, the system includes a computer network 202 to

5   be protected. Figure 14 also shows an exemplary intruder's system 220 such as might be used to attempt to gain access to the computer network 202. The computer network 202 comprises a plurality of network devices 204, an Internet access server 206, and a firewall 208 interposed between Internet access server 206 and the network connection to the Internet. Firewall 208 may be either a firewall, or a router with firewall functionality,

10   configured to route authorized users to Internet access server 206 and to detect and route unauthorized users to the trap system described below.

The system shown in Figure 14 also includes a trap system 1410, which comprises a trap host system 1412 in which multiple virtual cages 1414 are established. Also depicted are an administration console 1416 connected to trap host system 1412 for

15   allowing a system administrator to control the trap host system 1412 and the multiple virtual cages 1414 within it. Database 1418 within the trap system 1410 is used for storing data relating to activities within trap host system 1412 and virtual cages 1414.

The trap system 1410 is designed is designed to protect the computer network 202 from being accessed or otherwise compromised by an intruder (using intruder's system

20   220) who is attempting to gain access to computer network 202 via the Internet.

To facilitate establishment of multiple cages 1414 within the trap host system

1412, the trap host system 1412 is provided with multiple linecards (network interface

cards) 1502, as shown in Figure 15. It should be understood that five virtual cages 1414

have been shown for the sake of illustration, and that more or fewer cages 1414 may be

5   established within the trap host system 1412. In the embodiment shown in Figure 15,

each cage 1414 has a linecard 1502 associated with it, but may have more than one

linecard 1502. Because each linecard 1502 has an address associated with it, the trap

host system 1412 preferably has at least one linecard 1502 for each virtual cage 1414

established within the trap host system 1412, to facilitate creation and maintenance of the

10   cage environment and preventing the intruder from detecting that he is in a cage.

Figure 16 is a flow chart illustrating a process used in one embodiment to provide

computer security using a trap system such as trap system 1410 of Figure 14. The

process begins with step 1602 in which a trap system such as trap system 1410 of Figure

14 is installed. In step 1604, the file content for each deception environment to be

15   presented to would-be intruders is created, similar to step 304 in Figure 2. As before,

content may include fictitious content generated automatically as described above. The

number of deception environments may be determined in advance by the system

administrator, depending on the number of hosts he wishes to simulate (based on

perceived need and the ability of the trap host system to handle multiple cages, from the

20   standpoint of hardware resources, processing capability, memory, etc.), and a different

content set may be generated for each cage.

A trap is established within the trap system, step 1606. For example, multiple

virtual cages such as cages 1414 may be established within a trap host system, such as

trap host system 1412 of Figure 14. This is done by establishing file directories for the

cages and copying the operating system and the filesystem of the trap host system into

5   the directories, while omitting the modifications and additions to the operating system

that function to monitor the intruder's actions, keep the intruder in the cage, and prevent

the intruder from detecting that he is in a cage.

In step 1608, a would-be intruder is detected, in the same manner as described

above. A cage 1414 is selected according to the host to which the intruder is attempting

10   to gain access, step 1610. The cage 1414 may also be selected according to availability

(such as when multiple cages are being used to trap multiple intruders simultaneously),

the type of attack detected, the resource that the intruder is attempting to compromise, the

intruder's apparent identity/location, or other policies or criteria that the system

administrator wishes to implement. In step 1612, the would-be intruder is routed into the

15   trap system, such as trap system 1410 of Figure 14, as described herein, and directed to

the cage 1414 selected by the system according to policies and criteria determined by the

system administrator. Once the intruder has been routed into the trap, the intruder is kept

in the cage to ensure that the intruder does not break out of the trap system and gain

access to the portions of computer network 202 that are being protected from

20   unauthorized access, step 1614. In step 1616, the activity of the intruder within the trap

system is monitored. If the intruder attempts to attack a new host, step 1618, a cage

corresponding to the new host or new attack is selected, step 1620, and the intruder is routed to the new cage in the trap system, step 1612.

In step 1622, it is determined whether the intruder is leaving (activity has ceased). If not, the intruder is kept in the trap and selected cage, step 1614, and the intruder's activity is monitored, step 1616. If the intruder has ceased activity in the trap system, it is determined in step 1624 whether the changes made by the intruder to the configuration of the trap system during the attack should be kept in place. Alternatively, the determination of whether to keep changes could be made each time the intruder ceases activity in a cage (such as when opening a connection to a new host in step 1618) rather than only when the intruder ceases activity in the trap system. If it is determined in step 1624 that the changes will be kept, the process ends, leaving the state of the trap (or selected ones of the cages modified by the intruder) as modified by the intruder. If the changes will not be kept, the trap (or selected cages only) is reset to eliminate the changes made by the intruder, in step 1626. The trap may be reset to the state of the trap at the time the intruder entered, and may further have additional content generated to simulate ongoing usage by apparent users of the trap system so as to give the appearance of a computer system in normal operation.

Figure 17 illustrates a process flow in accordance with the invention, for use with multiple cages. In step 1702, the system call table (*sysent* in the exemplary trap system running Solaris 2.x) is instrumented. Modified functions are substituted for selected functions, to facilitate keeping the intruder in the cage without alerting the intruder to the fact that he is in a cage. The trap is set, as will be described below. In step 1704, an

intruder is detected and routed into the trap, and in step 1706, the intruder is assigned to a

cage, as described above. As the intruder is kept in the trap and selected cage and

intruder activity is monitored, system calls from inside the cage are intercepted. It is

determined whether the system call should be trapped, step 1708, and if not, the normal

5   system call is executed, step 1712. If the system call is one that would enable the

intruder to detect the presence of other cages or otherwise determine that he is in a cage, a

substitute system call is executed, step 1710. It should be understood that the above

process is given for the sake of illustration. The process may also be performed by

replacing appropriate functions in *sysent* and executing the substitute functions, which

10   will determine whether the system call should be handled normally or modified to

maintain the cage.

Figure 18 illustrates the process flow for setting the trap, as in step 1606 of Figure

16. In step 1802, multiple cages are established within the trap host system, such as by

creating new directories on the trap host system to contain the file structures for the

15   cages. The trap host system operating system is copied to each of the cages, step 1704.

As has been described herein, the interface to the operating system kernel is modified to

monitor the intruder's actions, keep the intruder in the cage, and prevent the intruder from

realizing that he is in the cage. Other modifications are made to facilitate the multiple

cage environment, as will be described below. The files and programs that perform these

20   functions are not copied into the cages, so that the intruder will not detect them.

Additionally, the filesystem of the trap host system is copied into the directories of the

cages, step 1806, and file content in the directories may be updated from time to time, to maintain the illusion of computer systems in normal use.

In one embodiment, the cages may be assigned to emulate hosts in the protected network, step 1808. Each cage may be created with file content and specific system resources (devices, network connections, etc.) to emulate a particular host that the system administrator wishes to establish as a decoy. The cages may also be configured to be used with specific types of attacks, or be configured similarly. When the intruder attempts to gain access to a host, the intruder will be routed into one of the cages, as has been described above.

As stated above, modifications to system calls in *sysent* are made to facilitate maintenance of the multiple cages, including preventing the intruder from detecting that he is in a cage. In one embodiment, *sysent* is modified to prevent the intruder from learning of the presence of other cages in the trap host system, by trapping calls that, for example, would allow the intruder to see or control processes outside of the cage, or give the intruder information about network connections that would enable him to detect that several apparent hosts are actually cages on a single system. This will be described in more detail below, by reference to some examples of modifications.

Figure 19 shows a process flow associated with a substituted *kill* function in *sysent*. In step 1902, the intruder, trapped in an assigned cage, causes a call to *kill* to be issued. The *kill* call is routed to the substituted *kill* function in *sysent*, which will be called *newkill* for the sake of convenience. It should be understood that in one

embodiment, the names for the substitute functions may be the same as the functions they replace; i.e. the substituted *kill* function would also be called *kill*. The *pid* passed to *kill* is used to determine whether the process the intruder is attempting to terminate is inside the current cage, step 1906. If the process is in the cage that the intruder is currently in, it is

5   terminated via execution of the normal *kill* function, as shown in step 1908. If the process is not in the current cage, the substituted *kill* function returns an ENOSUCHPROCESS error, step 1910. The implementation of *newkill* is illustrated in the following pseudo-code:

```
        newkill

10              if (in_the_cage)

                        return oldkill

                else

                        return NO_SUCH_PROCESS
```

If *kill* is invoked with the −1 flag (to kill all processes), *newkill* should kill only the

15   processes in the current cage.

The above-described modification of *kill* is to handle a situation in which an intruder might enter more than one cage simultaneously, and attempt to kill processes to test if the cages are on the same system. For example, if the intruder sees a list of processes in one cage (such as through *ps*) with associated process IDs, the intruder may

attempt to kill those processes from a second cage, even though those processes do not show up in a list of processes in the second cage.

To maintain the multiple cages undetected and cause them to appear to be different hosts, socket routines must be instrumented. The intruder should be prevented from testing the system's network connections to detect that he is in a cage or that two apparent hosts are actually the same host. The socket routines are used by the system to interact with the network, and must be modified to conceal true network information from being passed to the intruder and thereby alerting him to the presence of a cage. In short, the cages are made to appear to be different hosts with their own network connections. For example, in one embodiment, the trap host system might have five cages running. Most computers listen at 0.0.0.0 (IN_ADDR_ANY). If cage 1 listens at 0.0.0.0, it will receive all of the packets intended for the other cages. Thus, if an intruder were to telnet to cage 3, the intruder would get cage 1. To avoid this situation, each cage should have a linecard associated with it. This has the advantage of providing different hardware addresses and different routing tables as well, preventing the intruder from detecting the multiple cages via similar addresses and routing tables for supposedly different hosts.

Figure 20 illustrates an example of a *sysent* function that is instrumented in one embodiment of the invention. A call to *bind* is issued, step 2002, and this call is routed to the substitute *newbind* function in *sysent*, step 2004. In step 2006, the call is checked to determine whether it comes from inside the cage. If it does not, the original *bind* function (which has been termed *oldbind* for convenience) is invoked with *name* as the argument,

in step 2014. Otherwise, if the call originates in the cage, the process goes to step 2008, and determines whether *name* references the address of the cage, or localhost (0.0.0.0 or 127.0.0.1, by convention). If it does not, return error (ENOSUCHADDRESS), step 3010. If it does, the address of the cage is substituted for *name*, step 2012, and the original *bind*

5   function is invoked with the new *name* as the argument. This process may be written in pseudo-code as follows:

```
newbind

    if (not in cage)

        return oldbind

    elseif (name == 0.0.0.0 or 127.0.0.1 or cageaddr)

        name <- cageaddr

        return oldbind

    else

        return ENOSUCHADDRESS
```

15   One skilled in the art will readily recognize that the foregoing has the effect of changing *bind* so that when invoked with localhost as the argument, the actual *bind* will be invoked with the address of the cage. The error ENOSUCHADDRESS returned for other addresses would be expected, because the system should not be able to listen at other IP addresses.

Figure 21 illustrates a process flow in one embodiment of the invention for *listen*. Normally, *bind* is called first before *listen* is invoked for a socket. If *listen* is called without binding *name* first, the normal *listen* will default to running *bind* 0.0.0.0, which will choose a random port (above 1024). Thus, it is necessary to modify the *listen*

5 function as will be described. In step 2102, a call to *listen* is issued. The process checks whether *name* has been bound, step 2104. If *name* has been bound, the original *listen* function is called with *name* as the argument, as shown in step 2108. If *name* has not been bound, *newbind* is called with *name* set to 0.0.0.0, step 2106. The process proceeds to step 2108, calling *oldlisten* with *name* as the argument, step 2108. Pseudo-code for

10 this process may be written as follows:

newlisten

if (not bound)

newbind

return oldlisten

15   Figure 22 shows a process flow for one embodiment of the invention, for the *connect* system call. Typically, *connect* is called without binding first, and the system will choose any IP address available to it for the connection. Thus, *connect* must be modified to ensure that the IP address chosen is an address assigned to the cage in which the intruder is currently held. The process starts with a call to *connect*, issued with *name*

20 as the argument in step 2202. In step 2204, it is determined whether *name* has been

bound. If so, *oldconnect* is called with *name* as the argument, step 2208. Otherwise, *newbind* is called with *name* set to 0.0.0.0, step 2206, and *oldconnect* is called with the name as the argument, step 2208. The pseudo-code is as follows:

newconnect

5   if (not bound)

newbind

return oldconnect

In Figure 23, a process flow for *getsockname* is illustrated. If a socket has been flagged as being inside a cage, it may be rewritten with a value to make it work inside the cage. This value is concealed from the user/intruder in the cage, and the original value set by the intruder is stored. When the intruder invokes the *getsockname* routine, the instrumented routine returns the stored original value if the socket has been altered. As shown in Figure 23, a call to *getsockname* is issued, step 2302. The process checks whether *socket* has been renamed, step 2304. If *socket* has been renamed, the stored *oldname* (representing the original value of *socket* that was assigned in the cage) is returned, in step 2306. If *socket* has not been renamed, it is outside the cage and in step 2308, *oldgetsockname* is invoked with *socket* as the argument. This is illustrated by the following pseudo-code:

newgetsockname

20   if (renamed)

return oldname

else return oldgetsockname

As has been described above, modifications to functions that access the filesystem may be made in accordance with the invention, including modifications to

5   functions accessing *procfs* for hiding processes outside the cage. Similarly, modifications may be made to functions accessing *sockfs*. Interceptions of intruder-issued commands may be made with fine granularity to minimize the impact on system performance. An embodiment in accordance with the invention is shown in Figure 24. A call to *ioctl* that includes the arguments *cmd* and *fd* is issued in step 2402. This call is

10   routed to the substituted *ioctl* function placed in *sysent*, step 2404. The new *ioctl* function uses *fd* (file descriptor) to determine the type of *fs* and use the appropriate method. If *ioctl* is being performed on a socket, *ioctl* will run a *sockfs* method. Other types of *fs* will have their own methods.

In step 2408, *cmd* is extracted from the call to *ioctl* and the corresponding

15   modified function is executed in *newioctl*. For example, if *cmd* is *getnumif* (actually SIOCGIFNUM for get number of interfaces), the value 2 is returned, as shown in step 2410, because computers will have two interfaces, *lo0* (local) and *hme0* (network), and the intruder should not be informed of additional interfaces that might allow him to detect the presence of a cage or the fact that several apparent hosts are running on one computer

20   system. In step 2412, if *cmd* is *getifconfig* (which returns a list of interfaces), the list (*hme0, lo0*) is returned, for the same reasons as set forth above. If *cmd* is *getifaddr*

(*name*, such as *hme0*), step 2414, the process calls *oldioctl* with the name of the

corresponding real device, such as *qfe2*. If the call to *getifaddr* references a device not in

the cage, an error is returned. This process maintains each virtual cage and the interfaces

assigned to it, while concealing the existence of interfaces not associated with that cage.

5   The above *ioctl* modifications could be described in pseudo-code as follows:

newioctl (cmd, fd, …)

getnumif

return 2

getifconfig

10   return (hme0, lo0)

getifaddr (name)

oldioctl (name of real device)

The *netstat* system call, which shows the network status for the host, displays the

contents of various network-related data structures in various formats, depending on the

15   options specified in the call. This function is modified to control the information given to

the intruder, in order to prevent the intruder from detecting the presence of other cages on

the system. As shown in Figure 25, *netstat* sends a command (T_OPTMGMT_REQ) to

the various modules, such as TCP (Transmission Control Protocol), UDP (User Datagram

Protocol), ARP (Address Resolution Protocol), and IP (Internet Protocol). Depending on

the option specified in the call to *netstat*, each module lists its connections and other information, such as routing tables and network information. This information is reported to the stream head 2500, and is intercepted at the highest level before it is passed to the user/intruder. One skilled in the art will recognize that although the information is

5   intercepted at the stream head 2500 in the present embodiment, such information may be intercepted in other places, such as in the modules' reporting routines. The outputs of the modules are filtered to remove connections not associated with the cage that the intruder is currently in, in a manner similar to the filtering described above for *ufs* (Unix files) and *procfs* (processes).

10   For the sake of clarity, the system calls that may be modified in accordance with the invention have not been listed or described exhaustively, so that the present invention is not unnecessarily obscured. One skilled in the art will readily recognize that the concepts presented herein may be applied to other system calls, in order to implement the multiple cages described herein. Other commands that may be instrumented include

15   *sysinfo* and *ifconfig*. For example, *sysinfo*, which gets and returns system information strings, may be instrumented to return the expected hostname of the cage when invoked by an intruder inside the cage. The related *gethostname* system call may be modified in a similar manner.

        As has been described herein with reference to Figure 11A, a log of the intruder's

20   actions may be maintained, step 1102, and commands and/or keystrokes may be logged, along with information concerning the processes running within the cage or cages and other information. Over a period of time, the logfile can grow quite large, on the order of

many megabytes. Analyzing the logfile to discern actions and patterns is very time-consuming for a system administrator, and due to the sheer volume of information, it is fairly easy to miss intruder actions of significance. For example, there may be several actions that by themselves are of little significance, but taken together, may be interpreted

5   as an attempt to compromise the host system. There may be many intervening lines in the logfile, particularly when individual keystrokes are logged.

Therefore, in an embodiment of the invention, automated processing of the logfile is provided. Although keyword searches may be performed, this still results in many hits for the system administrator to examine, and such keyword searches are not sophisticated

10   enough to identify patterns and combinations that the system administrator may be interested in. In accordance with the invention, the logfile may be searched for patterns of interest, such as patterns corresponding to exploits and attacks, as well as general information about processes and system operations. Relevant lines may be highlighted for further analysis by the system administrator. The logfile may also be searched for

15   keystrokes that are aggregated and processed into a more readable format. In one embodiment, the logfile is searched using regular expressions. A regular expression is a pattern that describes a set of strings, and is constructed in a manner similar to an arithmetic expression, by using various operators to combine smaller expressions. Any regular expression can be represented as a context-free language.

20   Various programs exist for searching for regular expressions, such as *grep*, *fgrep*, *egrep*, and *perl*. Editors such as *vi* and *emacs* also support the use of regular expressions to find patterns in files. In one embodiment of the invention, queries are written in XML

and performed by an XML processor. In this manner, the system is configured to identify suspicious patterns and patterns of interest, as well as assist the system administrator in searching and analyzing the logfile.

In one embodiment, the system is configured to identify possible *sgid* (set group

5 ID) exploits. The XML for this query is illustrated in Figure 26. This query causes the system to look for the following pattern:

...exec args=...pid=(FOO1); ppid=...; uid=...; euid=...; gid=...;

egid=(0)...

followed by

10 ...args=...; pid=...; ppid=(FOO1)...

where the *gid* in the first line must start with a digit from 1-9. For each match that it finds, it will highlight the second line and alert the system administrator of a possible *sgid* exploit involving a child process of FOO1 (i.e., has a parent process ID of FOO1).

This looks for exploits involving using a *sgid* process, one that runs with *egid*

15 (effective group ID) equal to 0 (which allows the processes to run with root/super user privileges and gives them unlimited access to the resources of the host) but *gid* not equal to 0, to spawn another process. Normally, *sgid* processes should not spawn other processes. The system is configured to identify the spawned processes and highlight them for the system administrator.

The XML for detecting *suid* exploits in a logfile is similar, as illustrated in Figure 27. As shown in the figure, the query attempts to match the following:

...exec args=...pid=(FOO1); ppid=...; uid=...; euid=(0)...

followed by

5   ...args=...; pid=...; ppid=(FOO1)...

where the *uid* in the first line must start with a digit from 1-9. For each match found, the second line will be highlighted, and the system administrator will be alerted to a possible *suid* exploit involving a child process of FOO1. This identifies possible exploits involving setting *suid* (effective user ID) to 0 for root level access, and the spawned

10   processes are highlighted.

For example, this query would detect the following SUID root attack:

```
2001.02.21:12.49.11:96:rti.proclog: exec args=(./t4bin.compiled); pid=(7088);
ppid=(7037); uid=(100); euid=(100); gid=(1); egid=(1)

2001.02.21:12.49.12:96:rti.proclog: exec args=(rdist -d
```
15 `bleh=<A1>n<AC>t<A1>n<AC>t<A1>n<AC>t<A1>n<AC>t<A1>n<AC>t<A1>n<AC>t<A1>n<AC>t<A1>`
```
n<AC>t<A1>n<AC>t<A1>n<AC>t<A1>n<AC>t<A1>n<AC>t<A1>n<AC>t<A1>n<AC>t<A1>n<AC>t<A1
>n<AC>t<A1>); pid=(7088); ppid=(7037); uid=(100); euid=(0); gid=(1); egid=(1)

2001.02.21:12.49.12:128:rti.proclog: exec
args=(<AC>t<A1>n<AC>t<A1>n<AC>t<A1>n<AC>t<A1>n<AC>t<A1>n<AC>t<A1>n/bin/ksh<AC>t
```
20 `<A1>n<AC>t<A1>n<AC>t<A1>n<AC>t<A1>n<AC>t<A1>n<AC>t<A1>n<AC>t<A1>n<AC>t<A1>n<AC>`
```
t<A1>n<AC>t<A1>n<AC>t<A1>n<AC>t<A1>); pid=(7088); ppid=(7037); uid=(0);
euid=(0); gid=(1); egid=(1)
```

```
2001.02.21:12.49.12:128:rti.proclog: exec args=(/bin/pwd); pid=(7089);
ppid=(7088); uid=(0); euid=(0); gid=(1); egid=(1)
```

Note that rdist, an *suid* binary, causes a program to be executed as root.

The logfile may also be searched to find all log entries corresponding to processes

5   being executed, as shown by the query in Figure 28. This looks for lines that match the

following pattern:

…proclog…args=…

This is created in the logfile whenever a process is started. An example of a line

that would be matched is as follows:

10 ```
2001.02.15:13.47.03:128:rti.proclog: exec args=(/usr/bin/rm -f
/var/spool/lp/tmp/.net/requests/*/*); pid=(778); ppid=(777); uid=(0); euid=(0);
gid=(0); egid=(0)
```

Figure 29 shows a query for finding specific process. This query is configured to

ask for arguments from the system administrator (such as via the GUI), including

15  command line arguments, *pid, ppid, uid, euid, gid*, and *egid*. It finds all processes that

match that specification. For example, if given the argument pid=778, it would find:

```
2001.02.15:13.47.03:128:rti.proclog: exec args=(/usr/bin/rm -f
/var/spool/lp/tmp/.net/requests/*/*); pid=(778); ppid=(777); uid=(0); euid=(0);
gid=(0); egid=(0)
```

The system may also be configured to search for all processes spawned by a shell, as illustrated by the XML code in Figure 30. Lines that match the following pattern will be flagged as processes spawned from a shell:

...exec args=(-sh); pid=(FOO1)...

5   followed by

...args=...ppid=(FOO1)...

The query for tracking incoming connections is shown in Figure 31. Here, the query identifies all lines matching the following pattern:

...incoming connection from=...

10   and extracts the originating IP address, originating port, destination IP address, and destination port, which may then be displayed to the system administrator.

In one embodiment of the invention, individual keystrokes may be logged. This generates many lines of entries in the logfile, often interspersed with other entries such as those related to processes and connections. Thus, a query may be formulated to extract

15   and aggregate the keystrokes, as shown in Figure 32. This query finds all lines of the form:

```
2001.02.21:22.09.50:96:rti.strlog: read stream data, id=(0) data=(m)

2001.02.21:22.09.50:96:rti.strlog: read stream data, id=(0) data=(o)

2001.02.21:22.09.51:96:rti.strlog: read stream data, id=(0) data=(r)
```

```
2001.02.21:22.09.52:96:rti.strlog: read stream data, id=(0) data=(e\0a)
```

Upon receiving the newline character (\0a), the system will aggregate the keystrokes, and display "Keystrokes Entered: more\0a" in the GUI or other user interface.

The system may further be configured to track screen output recorded in the logfile, enabling a system administrator to see what information the intruder was displaying on his screen. Figure 33 illustrates the XML query for screen output, which searches for the pattern

...write stream data, id=(FOO1) data=...

followed by

...write stream data, id=(FOO1) data=...

As with the query for keystrokes, this query aggregates the output and displays it to the system administrator.

As has been described herein (e.g. with reference to Figure 9 and Figure 10), certain files may be monitored. These files may include system files, process files, network data files, and any other files considered sensitive. In one embodiment, the system is configured to track files opened, and the query is as shown in Figure 34 for matching lines in the logfile with the following pattern:

...monitored file opened name=(FILENAME) pid=FOO1...

After flagging a matching line in the logfile, the system may be configured to display the filename and *pid* of the process that opened the file, using the GUI as described above. It should be understood that although the embodiment described relates to opened files, one skilled in the art may configure the system to log attempted file accesses, and search for

5   the attempted file accesses as well.

For the sake of clarity, the processes and methods herein have been illustrated with a specific flow, but it should be understood that other sequences may be possible and that some may be performed in parallel, without departing from the spirit of the invention. Additionally, steps may be subdivided or combined. As disclosed herein,

10   software written in accordance with the present invention may be stored in some form of computer-readable medium, such as memory or CD-ROM, or transmitted over a network, and executed by a processor.

Although the foregoing invention has been described in some detail for purposes of clarity of understanding, it will be apparent that certain changes and modifications may

15   be practiced within the scope of the appended claims. It should be noted that there are many alternative ways of implementing both the process and apparatus of the present invention. Accordingly, the present embodiments are to be considered as illustrative and not restrictive, and the invention is not to be limited to the details given herein, but may be modified within the scope and equivalents of the appended claims.

20   WHAT IS CLAIMED IS: